JWT Security Report

Title: Security Analysis and Best Practices for JSON Web Tokens (JWT)

Prepared by: [SUNAY] ----------

1. Introduction

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. JWTs are widely used for authentication and authorization in modern applications, especially in Single Page Applications (SPAs) and API-based systems. While JWT simplifies stateless authentication, improper implementation can lead to severe security vulnerabilities. This report analyzes JWT security, common attack vectors, real-world exploitation methods, and mitigation strategies.

2. What is JWT?

A JWT is a Base64 URL-encoded string that consists of three parts:

Header.Payload.Signature

Header: Contains metadata, including the signing algorithm (e.g., HS256, RS256).

Payload: Contains claims (e.g., user ID, roles, expiration).

Signature: A cryptographic signature that verifies integrity and authenticity.

Example JWT:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

.eyJ1c2VySWQiOjEsIm5hbWUiOiJKb2huIERvZSJ9

.sflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

3. JWT Security Benefits

Stateless Authentication: No need to store session data on the server.

Compact and URL-safe: Easy to transmit via HTTP headers, query strings.

Cross-domain Support: Ideal for APIs and microservices.

4. Common JWT Vulnerabilities

4.1 Algorithm Confusion Attack

JWT supports multiple algorithms like HS256 (HMAC) and RS256 (RSA). Attackers can change alg in the header from RS256 to HS256 and sign using the public key asthe HMAC secret, bypassing verification.

4.2 None Algorithm Attack

Some libraries incorrectly allow alg: none, meaning no signature verification.

Attackers craft a token with "alg": "none" and modify payload freely.

4.3 Key Disclosure and Weak Secrets

Using short or guessable secrets for HS256 makes tokens brute-forceable.

4.4 Lack of Token Expiration

Tokens without exp claim or with long lifetimes remain valid indefinitely.

4.5 Insecure Storage on Client Side

Storing JWT in localStorage or sessionStorage exposes them to XSS attacks.

4.6 Token Replay Attack

If a JWT is leaked, it can be reused until it expires.

5. Real-World Attack Scenario (PoC)

Scenario: Exploiting None Algorithm Vulnerability

1. Capture a valid JWT.

2. Decode the header and change:

{ "alg": "none", "typ": "JWT" }

3. Modify payload (e.g., change role to admin). 4. Remove the signature part and send the token:

header.payload. 5. If the server doesn't validate signatures properly, you gain admin access. 6. JWT Security Best PracticesAlways enforce strong algorithms (HS256, RS256) and disallow none.

6. JWT Security Best PracticesAlways enforce strong algorithms (HS256, RS256) and disallow none.

Validate alg explicitly on the server side.Use strong secrets (minimum 256-bit forHMAC).Implement short token lifetimes (exp claim) and refresh tokens.Use HTTPS to preventMITM token theft.Avoid storing JWT in localStorage; prefer HTTP-only cookies.Implementtoken revocation (Blacklist or DB check on logout).Validate audience (aud), issuer (iss), andsubject (sub) claims.Rate-limit login attempts to prevent brute-force attacks.

7. Tools for JWT Testing

jwt.io → Decode & verify JWTs.

jwt_tool (Python) → Exploit JWT vulnerabilities. Burp Suite JWT Plugin → Security testing.

8. Conclusion

JWT provides a robust mechanism for stateless authentication, but improper implementation introduces severe vulnerabilities like algorithm confusion, none algorithm attacks, and token replay attacks. By following best practices, implementing proper validation, and securing token storage, developers can significantly reduce the attack surface.

9. References

RFC 7519-JSON Web Token (JWT

OWASP JWT Cheat Sheet

jwt. io